Name : VEDASHREE BHALERAO

Class : LY – AIA – 1

Batch : B

Roll No : 2213191

Assignment No. 2

Aim: Develop multi class classifier using deep multilayer perceptron (Keras/tensorflow/pytorch) for MNIST hand recognition dataset and CIFAR10. Fine the parameters for better accuracy.

- Develop application with GUI to upload input to the system
- Test the model **Objectives:**

- Learn Deep Neural Network modeling
- Learn to develop and deploy models **Theory:**

Standardisation

This is one of the most use type of scalar in data preprocessing . This is known as z-score . This re distribute the data in such a way that mean (μ) = 0 and standard deviation (σ) =1 . Here is the below formula for calculation

$$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation } (x)}$$

Normalization:

Normalization scales the feature between 0.0 & 1.0, retaining their proportional range to each other.

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

The range of normal distribution is [-1,1] with mean =0.

Data Splitting

Train Test Split is one of the important steps in Machine Learning. It is very important because your model needs to be evaluated before it has been deployed. And that evaluation needs to be done on unseen data because when it is deployed, all incoming data is unseen.

The main idea behind the train test split is to convert original data set into 2 parts

- train
- test where train consists of training data and training labels and test consists of testing data and testing labels.

The easiest way to do it is by using *scikit-learn*, which has a built-in function *train_test_split*

Data Cleaning

Data cleaning is the process of preparing data for analysis by removing or modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted.

This data is usually not necessary or helpful when it comes to analyzing data because it may hinder the process or provide inaccurate results. There are several methods for cleaning data depending on how it is stored along with the answers being sought.

Data cleaning is not simply about erasing information to make space for new data, but rather finding a way to maximize a data set's accuracy without necessarily deleting information.

For one, data cleaning includes more actions than removing data, such as fixing spelling and syntax errors, standardizing data sets, and correcting mistakes such as empty fields, missing codes, and identifying duplicate data points. Data cleaning is considered a foundational element of the data science basics, as it plays an important role in the analytical process and uncovering reliable answers.

Code :

- MNIST

```
import tensorflow as tf
from tensorflow.keras.models import Sequential from
tensorflow.keras.layers import Dense, Flatten, Dropout from
tensorflow.keras.utils import to_categorical from
tensorflow.keras.datasets import mnist import numpy as np
import tkinter as tk from tkinter import filedialog from
PIL import Image, ImageEnhance, ImageOps, ImageTk

# Load and preprocess MNIST data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
y_train = to_categorical(y_train, 10) y_test
= to_categorical(y_test, 10)

# Define the MNIST model mnist_model
= Sequential([
```

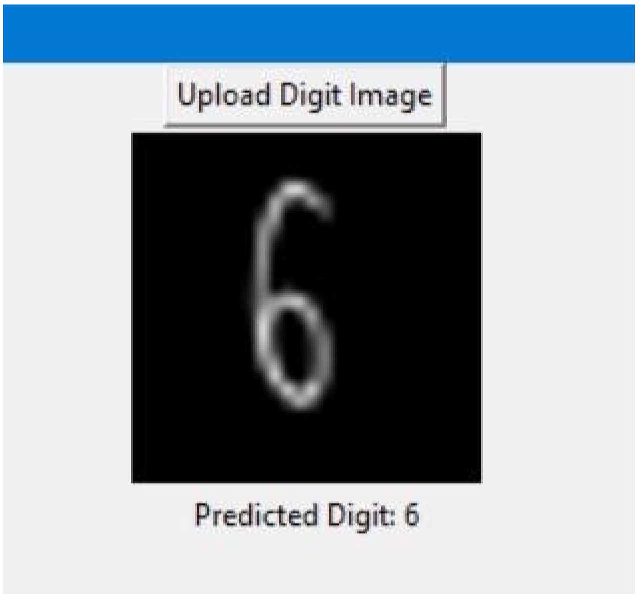
```
    Flatten(input_shape=(28, 28, 1)),  # Adding channel dimension
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(10, activation='softmax')
])  mnist_model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy']) mnist_model.fit(x_train[..., np.newaxis], y_train,
validation_data=(x_test[..., np.newaxis], y_test), epochs=10, batch_size=128)
# GUI function to upload and predict on MNIST model def
upload_and_predict_mnist():
    file_path = filedialog.askopenfilename()
if file_path:
        # Load image and ensure it's in grayscale              img =
Image.open(file_path).convert("L")  # Convert to grayscale
        # Enhance contrast to ensure the digit stands out
enhancer = ImageEnhance.Contrast(img)         img =
enhancer.enhance(2.0)

        # Resize to 28x28 and invert colors if necessary
img = img.resize((28, 28))
        img = ImageOps.invert(img) if np.array(img).mean() > 128 else img
        # Normalize and reshape for model input
img = np.array(img).astype("float32") / 255.0
        img = img.reshape(1, 28, 28, 1)  # Add channel dimension for grayscale
        # Make prediction and display result         prediction =
np.argmax(mnist_model.predict(img), axis=1)
result_label.config(text=f"Predicted Digit: {prediction[0]}")

# Tkinter GUI setup root
= tk.Tk()
root.title("Digit Recognition - MNIST")
upload_button = tk.Button(root, text="Upload Digit Image",
command=upload_and_predict_mnist)
upload_button.pack() result_label = tk.Label(root,
text="Prediction will appear here") result_label.pack()
root.mainloop()
```

Result :

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7570 - loss: 0.7800 - val_accuracy: 0.9511 - val_lo
ss: 0.1599
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9393 - loss: 0.2070 - val_accuracy: 0.9640 - val_lo
ss: 0.1139
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9569 - loss: 0.1491 - val_accuracy: 0.9707 - val_lo
ss: 0.0958
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9644 - loss: 0.1229 - val_accuracy: 0.9712 - val_lo
ss: 0.0899
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9673 - loss: 0.1074 - val_accuracy: 0.9759 - val_lo
ss: 0.0779
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9728 - loss: 0.0886 - val_accuracy: 0.9769 - val_lo
ss: 0.0744
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9748 - loss: 0.0834 - val_accuracy: 0.9773 - val_lo
ss: 0.0743
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9751 - loss: 0.0764 - val_accuracy: 0.9783 - val_lo
ss: 0.0714
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9773 - loss: 0.0702 - val_accuracy: 0.9804 - val_lo
ss: 0.0671
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9795 - loss: 0.0634 - val_accuracy: 0.9794 - val_lo
ss: 0.0700
```



- Cifar 10

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D from
tensorflow.keras.datasets import cifar10 from tensorflow.keras.utils
import to_categorical from tensorflow.keras.models import Sequential from
tensorflow.keras.layers import Dense, Flatten, Dropout from
tensorflow.keras.utils import to_categorical  # Add this import line
import numpy as np import tkinter as tk from tkinter import filedialog
from PIL import Image , ImageTk
```

```
# Load and preprocess CIFAR-10 data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype("float32") / 255.0 x_test =
x_test.astype("float32") / 255.0 y_train =
to_categorical(y_train, 10) y_test =
to_categorical(y_test, 10)

# Define the CIFAR-10 model cifar_model
= Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(10, activation='softmax')
])
cifar_model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
cifar_model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=50,
batch_size=64)
 cifar_labels = ["airplane", "automobile", "bird", "cat", "deer",
"dog", "frog", "horse", "ship", "truck"]

# GUI for uploading an image and testing the CIFAR-10 model def
upload_and_predict_cifar():
    file_path = filedialog.askopenfilename()
if file_path:
        #  Load  image  and  resize  to  32x32                    img   =
Image.open(file_path).convert("RGB")  # CIFAR-10 expects RGB images
        img_resized = img.resize((32, 32))

        # Display uploaded image on the screen
        img_display = ImageTk.PhotoImage(img_resized.resize((140, 140)))  #
Resize for better display in GUI
image_label.config(image=img_display)         image_label.image =
img_display

        # Normalize and reshape for model input          img_array =
np.array(img_resized).astype("float32") / 255.0         img_array =
img_array.reshape(1, 32, 32, 3)  # CIFAR-10 input shape
        # Make prediction and display result          prediction =
np.argmax(cifar_model.predict(img_array), axis=1)
result_label.config(text=f"Predicted Class:
```

```
{cifar_labels[prediction[0]]}")

# Tkinter GUI setup root = tk.Tk()
root.title("Image Recognition - CIFAR-10")

# GUI elements upload_button = tk.Button(root,
text="Upload Image",
command=upload_and_predict_cifar)
upload_button.pack()

# Label for displaying uploaded image
image_label = tk.Label(root) image_label.pack()

# Label for displaying prediction result
result_label = tk.Label(root, text="Prediction will appear here")
result_label.pack()

root.mainloop()
```

Result :

```
Epoch 1/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 23ms/step - accuracy: 0.3208 - loss: 1.8374 - val_accuracy: 0.5454 - val_
loss: 1.2883
Epoch 2/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 18s 23ms/step - accuracy: 0.5241 - loss: 1.3276 - val_accuracy: 0.6004 - val_
loss: 1.1321
Epoch 3/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 18s 23ms/step - accuracy: 0.5827 - loss: 1.1748 - val_accuracy: 0.6395 - val_
loss: 1.0315
Epoch 4/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 17s 22ms/step - accuracy: 0.6164 - loss: 1.0863 - val_accuracy: 0.6675 - val_
loss: 0.9626
Epoch 5/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 19s 25ms/step - accuracy: 0.6410 - loss: 1.0142 - val_accuracy: 0.6690 - val_
loss: 0.9603
```

```
Epoch 46/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - accuracy: 0.8193 - loss: 0.5043 - val_accuracy: 0.7445 - val_
loss: 0.8015
Epoch 47/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 19s 24ms/step - accuracy: 0.8168 - loss: 0.5019 - val_accuracy: 0.7414 - val_
loss: 0.8077
Epoch 48/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - accuracy: 0.8151 - loss: 0.5134 - val_accuracy: 0.7436 - val_
loss: 0.8071
Epoch 49/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - accuracy: 0.8216 - loss: 0.5000 - val_accuracy: 0.7385 - val_
loss: 0.8157
Epoch 50/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 25ms/step - accuracy: 0.8230 - loss: 0.4872 - val_accuracy: 0.7448 - val_
loss: 0.8142
```

Conclusion:

Thus, we have understood the syntax and basic model creation in TensorFlow for 2 different task.

We have also learned how to create a GUI using services to do so.